NVM Express Technical Errata

Errata ID	**027**
Change Date	**8/3/2012**
Affected Spec Ver.	**NVM Express 1.0c**
Corrected Spec Ver.	

Submission info

Name	Company	Date
Judy Brock	Samsung	4/26/2012
Ken Okin	Virident	4/26/2012
Dave Landsman	SanDisk	4/26/2012
Amber Huffman	Intel	4/26/2012
Peter Onufryk	IDT	4/26/2012
Don Walker	Dell	4/26/2012

This erratum attempts to use the pending command and candidate command language. More details to be added after feedback from the group.

Description of the specification technical flaw:

Modify the first two paragraphs of section 1.4 as shown below:

NVM Express is a scalable host controller interface designed to address the needs of Enterprise and Client systems that utilize PCI Express based solid state drives. The interface provides ~~an~~ optimized command ~~issue~~ submission and completion paths. It includes support for parallel operation by supporting up to 64K I/O Queues with up to 64K commands per I/O Queue. Additionally, support has been added for many Enterprise capabilities like end-to-end data protection (compatible with T10 DIF and SNIA DIX standards), enhanced error reporting, and virtualization.

The interface has the following key attributes:

- Does not require uncacheable / MMIO register reads in the command ~~issue~~ submission or completion path.
- A maximum of one MMIO register write is necessary in the command ~~issue~~ submission path.
- Support for up to 64K I/O queues, with each I/O queue supporting up to 64K commands.
- Priority associated with each I/O queue with well-defined arbitration mechanism.
- All information to complete a 4KB read request is included in the 64B command itself, ensuring efficient small I/O operation.
- Efficient and streamlined command set.
- Support for MSI/MSI-X and interrupt aggregation.
- Support for multiple namespaces.
- Efficient support for I/O virtualization architectures like SR-IOV.
- Robust error reporting and management capabilities.

Modify the sixth paragraph of section 1.4 as shown below:

An Admin Submission and associated Completion Queue exist for the purpose of device management and control (e.g., creation and deletion of I/O Submission and Completion Queues, aborting commands, etc.) Only commands that are part of the Admin Command Set may be ~~issued~~ submitted to the Admin Submission Queue.

Modify section 1.6.1 as shown below:

The Admin Queue is the Submission Queue and Completion Queue with identifier 0. The Admin Submission Queue and corresponding Admin Completion Queue are used to ~~issue~~ submit administrative commands and receive completions for those administrative commands, respectively.

The Admin Submission Queue is uniquely associated with the Admin Completion Queue.

Insert a new definition for command submission as section 1.6.4 as shown below:

1.6.4 candidate command

A candidate command is a submitted command the controller deems ready for processing.

Insert a new definition for command submission as section 1.6.6 as shown below:

1.6.6 command submission

A command is submitted when a Submission Queue Tail Doorbell write has completed that moves the Submission Queue Tail Pointer value past the corresponding Submission Queue entry for the associated command.

Modify bits 15:14 in the table in section 3.1.5 as shown below:

15:14	RW	0h	**Shutdown Notification (SHN):** This field is used to initiate shutdown processing when a shutdown is occurring, (i.e., a power down condition is expected.) For a normal shutdown notification, it is expected that the controller is given time to process the shutdown notification. For an abrupt shutdown notification, the host may not wait for shutdown processing to complete before power is lost. The shutdown notification values are defined as: (see table below) This field ~~Shutdown notification~~ should be ~~issued~~ written by host software prior to any power down condition and prior to any change of the PCI power management state. It is recommended that this field ~~shutdown notification~~ also be ~~sent~~ written prior to a warm reboot. To determine when shutdown processing is complete, refer to CSTS.SHST. Refer to section 7.6.2 for additional shutdown processing details.

Value	Definition
00b	No notification; no effect
01b	Normal shutdown notification
10b	Abrupt shutdown notification
11b	Reserved

Modify table in section 3.1.6 as shown below:

Bit	Type	Reset	Description
31:04	RO	0	Reserved
03:02	RO	0	**Shutdown Status (SHST):** This field indicates the status of shutdown processing that is initiated by the host setting the CC.SHN field. The shutdown status values are defined as: (see table below) To start executing commands on the controller after a shutdown operation (CSTS.SHST set to 10b), a reset (CC.EN cleared to '0') is required. If host software ~~issues~~ submits commands to the controller without issuing a reset, the behavior is undefined.
01	RO	0	**Controller Fatal Status (CFS):** This field is set to '1' when a fatal controller error occurred that could not be communicated in the appropriate Completion Queue. This field is cleared to '0' when a fatal controller error has not occurred. Refer to section **Error! Reference source not found.**.

Value	Definition
00b	Normal operation (no shutdown has been requested)
01b	Shutdown processing occurring
10b	Shutdown processing complete
11b	Reserved

Bit	Type	Reset	Description
00	RO	0	**Ready (RDY):** This field is set to '1' when the controller is ready to process commands after CC.EN is set to '1'. This field shall be cleared to '0' when CC.EN is cleared to '0'. Commands shall not be ~~issued~~ submitted to the controller until this field is set to '1' after the CC.EN bit is set to '1'. Failure to follow this requirement produces undefined results. Host software shall wait a minimum of CAP.TO seconds for this field to be set to '1' after setting CC.EN to '1' from a previous value of '0'.

Modify the first paragraph of section 3.1.10 as shown below:

This register defines the doorbell register that updates the Tail entry pointer for Submission Queue y. The value of y is equivalent to the Queue Identifier. This indicates to the controller that new commands have been submitted ~~are ready~~ for processing.

Modify the SQ Identifier field in Figure 13 as shown below:

Bit	Description
31:16	**SQ Identifier (SQID):** Indicates the Submission Queue ~~that~~ to which the associated command was submitted ~~issued to~~. This field is used by host software when more than one Submission Queue shares a single Completion Queue to uniquely determine the command completed in combination with the Command Identifier (CID).

Modify the Do Not Retry field in Figure 15 as shown below:

Bit	Description
31	**Do Not Retry (DNR):** If set to '1', indicates that if the same command is ~~re-issued~~ re-submitted it is expected to fail. If cleared to '0', indicates that the same command may succeed if retried. If a command is aborted due to time limited error recovery (refer to section 5.12.1.5), this field should be cleared to '0'.

Modify the Command Aborted due to SQ Deletion field in Figure 17 as shown below:

08h	**Command Aborted due to SQ Deletion:** The command was aborted due to a Delete I/O Submission Queue request received for the Submission Queue ~~that~~ to which the command was submitted ~~issued to~~.

Modify the fifth bullet of section 4.6 as shown below:

- If the host desires to abort the fused operation, the host shall submit ~~issue~~ an Abort command separately for each of the commands.

Modify section 4.7 as shown below:

4.7 Command Arbitration

~~A command is fetched when it is retrieved from host memory and stored internally to the controller. A command is launched when the controller begins executing that command.~~

~~Arbitration refers to the order in which commands submitted for execution by host software are launched for execution by the controller. The controller may access Submission Queues in any order. The controller fetches commands from memory for future execution in order from each individual Submission Queue it~~

~~accesses. The controller may store commands internally for future execution. Arbitration does not imply command completion order, rather arbitration determines the order in which commands that are launched for execution by the controller. Since commands are of different types, of different sizes, and to different LBA ranges on the controller, the order of completion is likely to be different than the order of command launch.~~

A command is submitted when a Submission Queue Tail Doorbell write moves the Submission Queue Tail Pointer past the corresponding Submission Queue entry. The controller transfers submitted commands to the controller's local memory for subsequent processing using a vendor specific algorithm.

A command is being processed when the controller and/or namespace state is being accessed or modified by the command (e.g., a Feature setting is being accessed or modified or a logical block is being accessed or modified).

A command is complete when a Completion Queue entry for the command has been posted to the corresponding Completion Queue. Upon completion, all controller state and/or namespace state modifications made by that command are globally visible to all subsequently submitted commands.

A candidate command is a submitted command the controller deems ready for processing. The controller selects command(s) for processing from the pool of submitted commands for each Submission Queue. The commands that comprise a fused operation shall be processed together and in order by the controller. The controller may select candidate commands for processing in any order. The order in which commands are selected for processing does not imply the order in which commands are completed.

Arbitration is the method used to determine the Submission Queue from which the controller will start processing the next candidate command(s). Once a Submission Queue is selected using arbitration, the Arbitration Burst setting determines the maximum number of commands that the controller may start processing from that Submission Queue before arbitration shall again take place. A fused operation may be considered as one or two commands by the controller.

All controllers shall support the round robin command arbitration mechanism. A controller may optionally implement weighted round robin with urgent priority class and/or a vendor specific arbitration mechanism. The Arbitration Mechanism Supported field in the Controller Capabilities register (CC.AMS) indicates optional arbitration mechanisms supported by the controller.

~~A command is ready for execution when a Submission Queue Tail Doorbell write has completed that moves the Submission Queue Tail Pointer value past the corresponding Submission Queue entry for the associated command. Within the same Submission Queue, ready commands may be launched in any order.~~

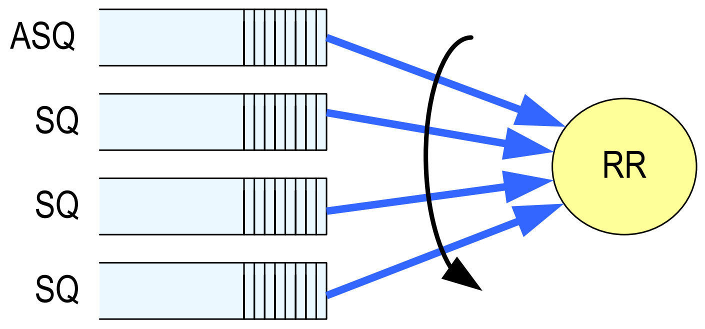
In order to make efficient use of the non-volatile memory, it is often advantageous to execute multiple commands from a Submission Queue in parallel. For Submission Queues that are using weighted round robin with urgent priority class or round robin arbitration, host software may configure an Arbitration Burst setting. The Arbitration Burst setting indicates the maximum number of commands that the controller may launch at one time from a particular Submission Queue. It is recommended that host software configure the Arbitration Burst setting as close to the recommended value by the controller as possible (specified in the Recommended Arbitration Burst field of the Identify Controller data structure in Figure 66), taking into consideration any latency requirements. Refer to section 5.12.1.1.

~~If required resources (e.g. logical block locations) are not available that command(s) require, then those command(s) may be deferred for launch to the next arbitration round. In some cases, this may result in fewer commands being launched from a particular Submission Queue in an arbitration round.~~

4.7.1 Round Robin Arbitration

If the round robin arbitration mechanism is selected, the controller shall implement round robin command arbitration amongst all Submission Queues, including the Admin Submission Queue. In this case, all Submission Queues are treated with equal priority. The controller may ~~launch~~ select multiple candidate commands for processing from each Submission Queue per round based on the Arbitration Burst setting.

Figure 23: Round Robin Arbitration



4.7.2 Weighted Round Robin with Urgent Priority Class Arbitration

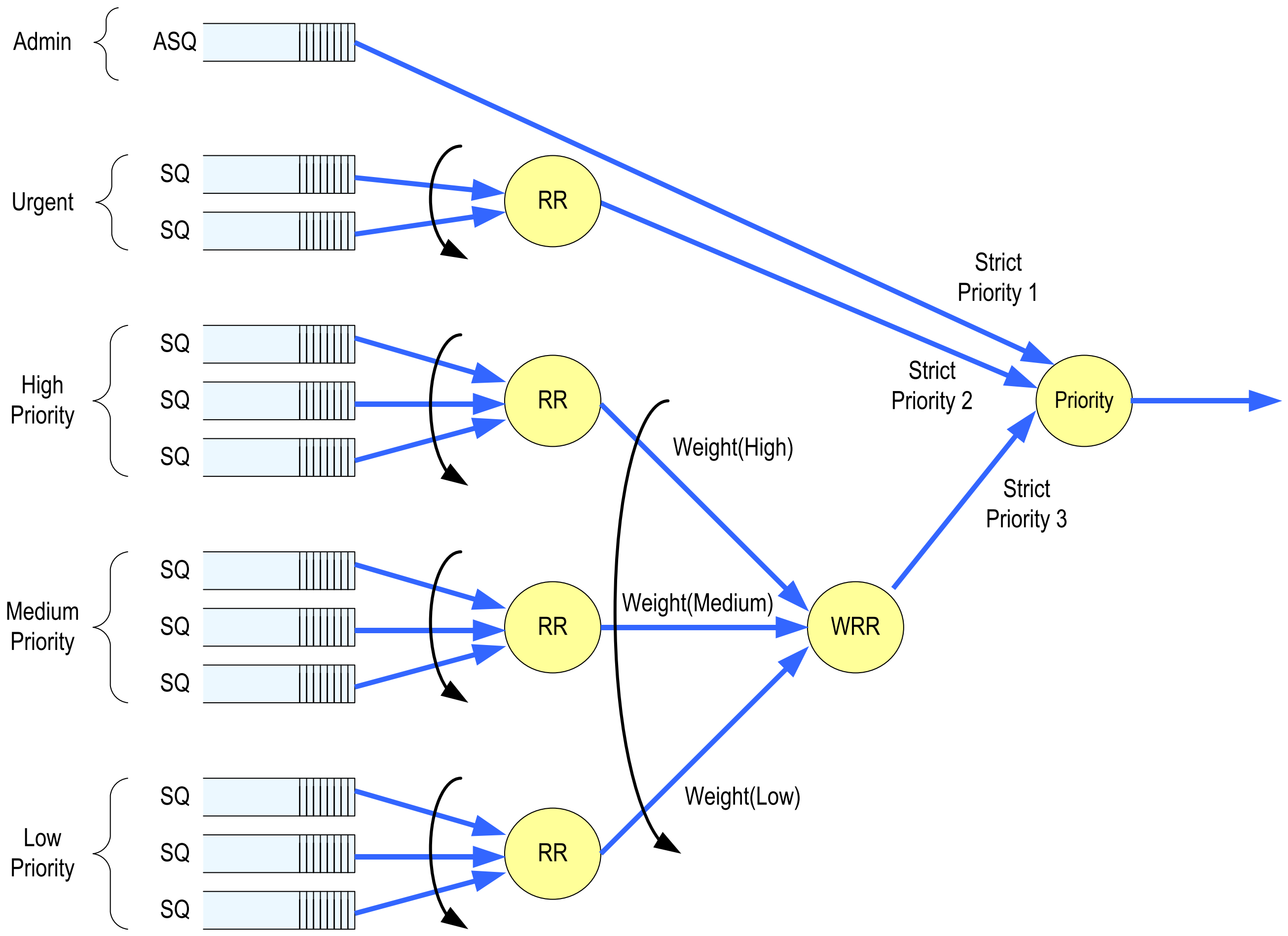
In this arbitration mechanism, there are three strict priority classes and three weighted round robin priority levels. If Submission Queue A is of higher strict priority than Submission Queue B, then all candidate commands ~~that are ready for execution~~ in Submission Queue A shall start processing ~~be launched~~ before candidate commands from Submission Queue B start processing ~~are launched~~.

The highest strict priority class is the Admin class that includes any command submitted ~~issued~~ to the Admin Submission Queue. This class has the highest strict priority above commands submitted ~~issued~~ to any other Submission Queue.

The next highest strict priority class is the Urgent class. Any I/O Submission Queue assigned to the Urgent priority class is serviced next after commands submitted ~~issued~~ to the Admin Submission Queue, and before any commands submitted ~~issued~~ to a weighted round robin priority level. Host software should use care in assigning any Submission Queue to the Urgent priority class since there is the potential to starve I/O Submission Queues in the weighted round robin priority levels as there is no fairness protocol between Urgent and non Urgent ~~with those~~ I/O Submission Queues.

The lowest strict priority class is the Weighted Round Robin class. This class consists of the three weighted round robin priority levels (High, Medium, and Low) that share the remaining bandwidth using weighted round robin arbitration. Host software controls the weights for the High, Medium, and Low service classes via Set Features. Round robin is used to arbitrate within multiple Submission Queues assigned to the same weighted round robin level. The number of candidate commands that may start processing ~~be launched~~ from each Submission Queue per round is either the Arbitration Burst setting or the remaining weighted round robin credits, whichever is smaller.

Figure 24: Weighted Round Robin with Urgent Priority Class Arbitration



In Figure 24, the Priority decision point selects the highest priority candidate command selected next to start processing ~~submitted for execution next~~.

Modify the first paragraph of section 5 as shown below:

The Admin Command Set defines the commands that may be ~~issued~~ submitted to the Admin Submission Queue.

Modify the first paragraph of section 5.1 as shown below:

The Abort command is used to abort a specific command previously ~~issued~~ submitted to the Admin Submission Queue or an I/O Submission Queue. Host software may have multiple Abort commands outstanding, subject to the constraints of the Abort Command Limit indicated in the Identify Controller data structure in Figure 66. An Abort command is a best effort command; the command to abort may have already completed, currently be in execution, or may be deeply queued. It is implementation specific if/when a controller chooses to complete the command when the command to abort is not found.

Modify the first four paragraphs of section 5.2 as shown below:

Asynchronous events are used to notify host software of error and health information as these events occur. To enable asynchronous events to be reported by the controller, host software needs to ~~issue~~ submit one or more Asynchronous Event Request commands to the controller. The controller indicates an event to the host by completing an Asynchronous Event Request command. Host software should expect that the controller may not execute the command immediately; the command should be completed when there is an event to be reported.

The Asynchronous Event Request command is ~~issued~~ submitted by host software to enable the reporting of asynchronous events from the controller. This command has no timeout. The controller posts a completion queue entry for this command when there is an asynchronous event to report to the host. If Asynchronous Event Request commands are outstanding when the controller is reset, the commands are aborted.

All command specific fields are reserved.

Host software may ~~issue~~ submit multiple Asynchronous Event Request commands to reduce event reporting latency. The total number of simultaneously outstanding Asynchronous Event Request commands is limited by the Asynchronous Event Request Limit specified in the Identify Controller data structure in Figure 66.

Modify the second paragraph of section 5.6 as shown below:

The command causes all commands ~~issued~~ submitted to the indicated Submission Queue that are still in progress to be aborted. The controller does not need to post individual completion status for commands that have been aborted. Commands that are not able to be aborted should be completed with appropriate completion status.

Modify the first paragraph of section 5.6.1 as shown below:

After all commands ~~issued~~ submitted to the indicated I/O Submission Queue are either completed or aborted, a completion queue entry is posted to the Admin Completion Queue when the queue has been deleted. The completion queue entry shall indicate if commands were aborted. Delete I/O Submission Queue command specific status values are defined in Figure 44.

Modify the first two paragraphs of section 5.8 as shown below:

The Firmware Image Download command is used to download all or a portion of the firmware image for a future update to the controller. The Firmware Image Download command may be ~~issued~~ submitted while other commands on the Admin Submission Queue or I/O Submission Queues are outstanding. The Firmware Image Download command copies the new firmware image (in whole or in part) to the controller.

The firmware image may be constructed of multiple pieces that are individually downloaded with separate Firmware Image Download commands. Each Firmware Image Download command includes a Dword Offset and Number of Dwords that specify a Dword range. The host software shall ensure that firmware pieces do not have Dword ranges that overlap. Firmware portions may be ~~issued~~ submitted out of order to the controller.

Modify Figure 60 as shown below:

Figure 60: Get Log Page – SMART / Health Information Log

Bytes	Description
0	**Critical Warning:** This field indicates critical warnings for the state of the controller. Each bit corresponds to a critical warning type; multiple bits may be set. If a bit is cleared to '0', then that critical warning does not apply. Critical warnings may result in an asynchronous event notification to the host. **Bit** / **Definition** 00: If set to '1', then the available spare space has fallen below the threshold. 01: If set to '1', then the temperature has exceeded a critical threshold. 02: If set to '1', then the device reliability has been degraded due to significant media related errors or any internal error that degrades device reliability. 03: If set to '1', then the media has been placed in read only mode. 04: If set to '1', then the volatile memory backup device has failed. This field is only valid if the controller has a volatile memory backup solution. 07:05: Reserved
2:1	**Temperature:** Contains the temperature of the overall device (controller and NVM included) in units of Kelvin. If the temperature exceeds the temperature threshold, refer to section **Error! Reference source not found.**, then an asynchronous event completion may occur ~~may be issued to the host~~.
3	**Available Spare:** Contains a normalized percentage (0 to 100%) of the remaining spare capacity available.
4	**Available Spare Threshold:** When the Available Spare falls below the threshold indicated in this field, an asynchronous event completion may occur ~~may be issued to the host~~. The value is indicated as a normalized percentage (0 to 100%).
5	**Percentage Used:** Contains a vendor specific estimate of the percentage of device life used based on the actual device usage and the manufacturer's prediction of device life. A value of 100 indicates that the estimated endurance of the device has been consumed, but may not indicate a device failure. The value is allowed to exceed 100. Percentages greater than 254 shall be represented as 255. This value shall be updated once per power-on hour (when the controller is not in a sleep state). Refer to the JEDEC JESD218 standard for SSD device life and endurance measurement techniques.
31:6	Reserved
47:32	**Data Units Read:** Contains the number of 512 byte data units the host has read from the controller; this value does not include metadata. This value is reported in thousands (i.e., a value of 1 corresponds to 1000 units of 512 bytes read) and is rounded up. When the LBA size is a value other than 512 bytes, the controller shall convert the amount of data read to 512 byte units. For the NVM command set, logical blocks read as part of Compare and Read operations shall be included in this value.
63:48	**Data Units Written:** Contains the number of 512 byte data units the host has written to the controller; this value does not include metadata. This value is reported in thousands (i.e., a value of 1 corresponds to 1000 units of 512 bytes written) and is rounded up. When the LBA size is a value other than 512 bytes, the controller shall convert the amount of data written to 512 byte units. For the NVM command set, logical blocks written as part of Write operations shall be included in this value. Write Uncorrectable commands shall not impact this value.

79:64	**Host Read Commands:** Contains the number of read commands ~~issued to~~ completed by the controller. For the NVM command set, this is the number of Compare and Read commands.
95:80	**Host Write Commands:** Contains the number of write commands ~~issued to~~ completed by the controller. For the NVM command set, this is the number of Write commands.
111:96	**Controller Busy Time:** Contains the amount of time the controller is busy with I/O commands. The controller is busy when there is a command outstanding to an I/O Queue (specifically, a command was ~~issued~~ submitted via an I/O Submission Queue Tail doorbell write and the corresponding completion queue entry has not been posted yet to the associated I/O Completion Queue). This value is reported in minutes.
127:112	**Power Cycles:** Contains the number of power cycles.
143:128	**Power On Hours:** Contains the number of power-on hours. This does not include time that the controller was powered and in a low power state condition.
159:144	**Unsafe Shutdowns:** Contains the number of unsafe shutdowns. This count is incremented when a shutdown notification (CC.SHN) is not received prior to loss of power.
175:160	**Media Errors:** Contains the number of occurrences where the controller detected an unrecovered data integrity error. Errors such as uncorrectable ECC, CRC checksum failure, or LBA tag mismatch are included in this field.
191:176	**Number of Error Information Log Entries:** Contains the number of Error Information log entries over the life of the controller.
511:192	Reserved

Modify byte 77 of Figure 66 as shown below:

77	M	**Maximum Data Transfer Size (MDTS):** This field indicates the maximum data transfer size between the host and the controller. The host should not ~~issue~~ submit a command that exceeds this transfer size. If a command is ~~processed~~ submitted that exceeds the transfer size, then the command is aborted with a status of Invalid Field in Command. The value is in units of the minimum memory page size (CAP.MPSMIN) and is reported as a power of two (2^*n*). A value of 0h indicates no restrictions on transfer size. The restriction includes metadata if it is interleaved with the logical block data.

Modify bytes 525 through 529 of Figure 66 as shown below:

525	M	**Volatile Write Cache (VWC):** This field indicates attributes related to the presence of a volatile write cache in the implementation. Bits 7:1 are reserved. Bit 0 if set to ‘1’ indicates that a volatile write cache is present. If cleared to ‘0’, a volatile write cache is not present. If a volatile write cache is present, then the host may issue Flush commands and control whether it is enabled with Set Features specifying the Volatile Write Cache feature identifier. If a volatile write cache is not present, the host shall not ~~issue~~ submit Flush commands nor Set Features or Get Features with the Volatile Write Cache identifier.
527:526	M	**Atomic Write Unit Normal (AWUN):** This field indicates the atomic write size for the controller during normal operation. This field is specified in logical blocks and is a 0’s based value. If a write is ~~issued~~ submitted of this size or less, the host is guaranteed that the write is atomic to the NVM with respect to other read or write operations. A value of FFFFh indicates all commands are atomic as this is the largest command size. It is recommended that implementations support a minimum of 128KB (appropriately scaled based on LBA size).

529:528	M	**Atomic Write Unit Power Fail (AWUPF):** This field indicates the atomic write size for the controller during a power fail condition. This field is specified in logical blocks and is a 0's based value. If a write is ~~issued~~ submitted of this size or less, the host is guaranteed that the write is atomic to the NVM with respect to other read or write operations.

Modify the second paragraph of section 5.12.1 as shown below:

There may be commands in execution when a Feature is changed. The new settings may or may not apply to commands already submitted for execution when the Feature is changed. Any commands ~~issued~~ submitted to a Submission Queue after a Set Features is successfully completed shall utilize the new settings for the associated Feature. To ensure that a Feature applies to all subsequent commands, commands being processed ~~in execution~~ should be completed prior to issuing the Set Features command.

Modify the first sentence before the comma in the second paragraph of section 5.12.1.1, 5.12.1.2, 5.12.1.3, 5.12.1.4, 5.12.1.5, 5.12.1.6, 5.12.1.7, 5.12.1.8, 5.12.1.9, 5.12.1.10, 5.12.1.11, and 5.12.1.12 as shown below:

If a Get Features command is ~~issued~~ submitted for this Feature,

Modify the first paragraph of section 5.14 as shown below:

The Security Receive command transfers the status and data result of one or more Security Send commands that were previously ~~issued~~ submitted to the controller.

Modify the first paragraph of section 5.15 as shown below:

The Security Send command is used to transfer security protocol data to the controller. The data structure transferred to the controller as part of this command contains security protocol specific commands to be performed by the controller. The data structure transferred may also contain data or parameters associated with the security protocol commands. Status and data that is to be returned to the host for the security protocol commands ~~issued~~ submitted by a Security Send command are retrieved with the Security Receive command defined in section 5.14.

Modify the second paragraph of section 6 as shown below:

The NVM Command Set includes the commands listed in Figure 99. The following subsections describe the definition for each of these commands. Commands shall only be ~~issued~~ submitted by the host when the controller is ready as indicated in the Controller Status register (CSTS.RDY) and after appropriate I/O Submission Queue(s) and I/O Completion Queue(s) have been created.

Modify section 6.3 as shown below:

Except for commands that are part of a fused operation, each command is processed as an independent entity without reference to other commands ~~issued~~ submitted to the same I/O Submission Queue or to commands ~~issued~~ submitted to other I/O Submission Queues. Specifically, the controller is not responsible for checking the LBA of a Read or Write command to ensure any type of ordering between commands. For example, if a Read is ~~issued~~ submitted for LBA x and there is a Write also ~~issued~~ submitted for LBA x, there is no guarantee of the order of completion for those commands (the Read may finish first or the Write may finish first).

If there are ordering requirements between commands, host software or the associated application is required to enforce that ordering above the level of the controller.

The controller supports an atomic write unit. The atomic write unit is the size of write operation guaranteed to be written atomically to the medium with respect to other read or write operations. The controller supports a value for normal operation that is potentially different than during a power fail condition, as reported in the Identify Controller data structure. The host may indicate that the atomic write unit beyond a logical block size is not necessary by configuring the Write Atomicity feature, which may result in higher performance in some implementations.

After a write has completed, reads for that location ~~that~~ which are subsequently ~~complete~~ submitted shall return the data from that write and not an older version of the data from a previous write.

Modify section 7.1 as shown below:

Host software ~~presents~~ submits commands to the controller through pre-allocated Submission Queues. The controller is alerted to ~~new commands to execute~~ newly submitted commands through SQ Tail Doorbell register writes. The difference between the previous doorbell register value and the current register write indicates the number of commands that were submitted ~~are ready for processing by the controller~~.

The controller fetches the commands from the Submission Queue(s) and transmits them to the NVM subsystem for processing. Except for fused operations, there are no ordering restrictions for processing of the commands within or across Submission Queues. Host software should not place commands in the list that may not be re-ordered arbitrarily. Data may or may not be committed to the NVM media in the order that commands are received.

Host software ~~issues~~ submits commands of higher priorities to the appropriate Submission Queues. Priority is associated with the Submission Queue itself, thus the priority of the command is based on the Submission Queue it is issued through. The controller arbitrates across the Submission Queues based on fairness and priority according to the arbitration scheme specified in section 4.7.

Upon completion of the commands by the NVM subsystem, the controller presents completion queue entries to the host through the appropriate Completion Queues. If MSI-X or multiple message MSI is in use, then the interrupt vector indicates the Completion Queue(s) with possible new command completions for the host to process. If pin-based interrupts or single message MSI interrupts are used, host software interrogates the Completion Queue(s) for new completion queue entries. The host updates the CQ Head doorbell register to release Completion Queue entries to the controller and clear the associated interrupt.

There are no ordering restrictions for completions to the host. Each completion queue entry identifies the Submission Queue Identifier and Command Identifier of the associated command. Host software uses this information to correlate the completions with the commands ~~issued~~ submitted to the Submission Queue(s).

Host software is responsible for creating all required Submission and Completion Queues prior to ~~issuing~~ submitting commands to the controller. I/O Submission and Completion Queues are created using Admin commands defined in section 5.

Modify section 7.2 as shown below:

7.2 Command ~~Issue~~ Submission and Completion Mechanism (Informative)

This section describes the command ~~issue~~ submission and completion mechanism. It also describes how commands are built by host software and command completion processing.

Modify the first paragraph and bullets 1 and 2 of section 7.2.1 as shown below:

This section describes command ~~issue~~ submission and completion processing. Figure 137 shows the steps that are followed to issue and complete a command. The steps are:

1. The host creates a command for execution within the appropriate Submission Queue in host memory.
2. The host updates the Submission Queue Tail Doorbell register with the new value of the Submission Queue Tail entry pointer. This indicates to the controller that a new command(s) is ~~ready~~ submitted for processing.

Modify bullet 2 of section 7.2.2 as shown below:

2. Host software shall write the corresponding Submission Queue doorbell register (SQxTDBL) to submit ~~indicate to the controller that~~ one or more commands ~~are available~~ for processing.

Modify the next to last paragraph of section 7.2.5.1 as shown below:

After the command is built, host software ~~issues~~ submits the command for execution by writing the Admin Submission Queue doorbell (SQ0TDBL) to indicate to the controller that this command is available for processing.

Modify the last paragraph of section 7.2.5.2 as shown below:

After the commands are built, host software ~~issues~~ submits the commands for execution by writing the appropriate I/O Submission Queue doorbell (SQxTDBL) to indicate to the controller that these commands are ~~available for processing~~ submitted. Note that the doorbell write shall indicate both commands ~~are available~~ have been submitted at one time.

Modify section 7.3.2 as shown below:

The host may reset and/or reconfigure the Submission and Completion Queues by resetting them. A queue level reset is performed by deleting and then recreating the queue. In this process, the host should wait for all pending commands to the appropriate Submission Queue(s) to complete. To perform the reset, the host ~~issues~~ submits the Delete I/O Submission Queue or Delete I/O Completion Queue command to the Admin Queue specifying the identifier of the queue to be deleted. After successful command completion of the queue delete operation, the host then recreates the queue by ~~issuing~~ submitting the Create I/O Submission Queue or Create I/O Completion Queue command. As part of the creation operation, the host may modify the attributes of the queue if desired.

The host should ensure that the appropriate Submission Queue or Completion Queue is idle before deleting it. ~~Issuing~~ Submitting a queue deletion command causes any pending commands to be aborted by the controller; this may or may not result in a completion queue entry being posted for the aborted command(s). Note that if a queue level reset is performed on a Completion Queue, the Submission Queues that are utilizing the Completion Queue should be reset as part of the same operation. The behavior of a Submission Queue without a corresponding Completion Queue is undefined.

Modify bullet 2 of section 7.4.1 as shown below:

2. ~~Issues~~ Submits a Set Features command for the Number of Queues attribute in order to request the number of I/O Submission Queues and I/O Completion Queues desired. The completion of this Set

Features command indicates the number of I/O Submission Queues and I/O Completion Queues allocated.

Modify section 7.4.3 as shown below:

To abort a large number of commands, the recommended procedure is to delete and recreate the I/O Submission Queue. Specifically, to abort all commands that are ~~issued~~ submitted to the Submission Queue host software should issue a Delete I/O Submission Queue command for that queue. After the queue has been successfully deleted, indicating that all commands have been completed or aborted, then host software should recreate the queue by ~~issuing~~ submitting a Create I/O Submission Queue command. Host software may then ~~re-issue~~ re-submit any commands desired to the associated I/O Submission Queue.

Modify section 7.6.1 as shown below:

10. If the host desires asynchronous notification of error or health events, the host should ~~issue~~ submit an appropriate number of Asynchronous Event Request commands. This step may be done at any point after the controller signals it is ready (i.e., CSTS.RDY is set to ‘1’).

Modify the first bullet 1 of section 7.6.2 as shown below:

1. Stop ~~issuing~~ submitting any new I/O commands to the controller and allow any outstanding commands to complete.

Modify the second bullet 1 of section 7.6.2 as shown below:

1. Stop ~~issuing~~ submitting any new I/O commands to the controller.

Modify bullet 2 of section 8.1 as shown below:

2. After the firmware is downloaded to the controller, the next step is for the host to ~~issue~~ submit a Firmware Activate command. The Firmware Activate command verifies that the last firmware image downloaded is valid and commits that image to the firmware slot indicated for future use. A firmware image that does not start at offset zero, contains gaps, or contains overlapping regions is considered invalid. A controller may employ additional vendor specific means (e.g., checksum, CRC, cryptographic hash or a digital signature) to determine the validity of a firmware image.
    a. The Firmware Activate command may also be used to activate a firmware image associated with a previously activated firmware slot.

Modify the fourth paragraph of section 8.4 as shown below:

The host may dynamically modify the power state using the Set Features command and determine the current power state using the Get Features command. The host may directly transition between any two supported power states. The Entry Latency (ENTLAT) field in the power management descriptor indicates the maximum amount of time in microseconds that it takes to enter that power state and the Exit Latency (EXTLAT) field indicates that maximum amount of time in microseconds that it takes to exit that state. The maximum amount of time to transition between any two power states is equal to the sum of the old state’s exit latency and the new state’s entry latency. The host is not required to wait for a previously ~~issued~~ submitted power state transition to complete before initiating a new transition. The maximum amount of time for a sequence of power state transitions to complete is equal to the sum of transition times for each individual power state transition in the sequence.

Modify section 9.4 as shown below:

Device specific errors such as a DRAM failure or power loss notification errors indicate that a controller level failure has occurred during the processing of a command. The status code of the completion queue entry should indicate an Internal Device Error status code (if multiple error conditions exist, the lowest numerical value is returned). Host software shall ignore any data transfer associated with the command. The host may choose to ~~re-issue~~ re-submit the command or indicate an error to the higher level software.

Disposition log

3/14/2012	Erratum started.
3/21/2012	Added changes to subsections 4.7.1 and 4.7.2.
4/18/2012	Updates from 4/5 meeting and reflector to first paragraph of 4.7.
5/30/2012	Added Arbitration Burst clarification.
6/13/2012	Added fused command clarification and chapter 1 updates.
6/21/2012	Added chapter 3 updates.
6/27/2012	Added Ken's proposed change to candidate wording, added chapter 4 updates.
7/18/2012	Added HGST's command processing definition update; added chapter 4 and 5 updates.
7/24/2012	Finished capturing Ken's changes.
8/3/2012	Updated section 4.7 based on 8/2 Workgroup meeting discussion.
9/10/2012	ECN ratified.